Appendix

Contents

1.0 Introduction

This report is for CSE228 project. I picked the first project from the PDF. Here is the requirements from it.

Healthcare Management System
➢ **Description:** A system for managing the data of doctors and patients ▪ There are 2 types of patients: **Emergency patient** and **Normal patient** ▪ Each Patient "normal and emergency" has **ID**, **name**, **address**, **phone number**, **gender**, **symptoms**, **payment method** and **diagnosis**. And in the case of emergency patient there is **room number**. ▪ Each doctor has **ID**, **name**, **address**, **phone number**, and **specialty in medicine**. ❖ This application should enable the admin to: ▪ Add new doctor ▪ Add new patient ▪ Edit existing doctor ▪ Edit existing patient ▪ Delete doctor ▪ Delete patient ▪ Display all doctors ▪ Display emergency patients and normal patients ▪ Make an appointment for a patient with specific doctor ➢ [Bonus] ▪ Using files or database to manage the data in all the required functions ▪ Extra complicated functionalities ▪ Powerful GUI

So, I made up my mind and I tried to take this project seriously and professionally. I did not implement any security feature. But the program - in the end - is fully functional and has a **powerful GUI** with lots of small tweaks and a **database** that saves and loads locally.

I will talk thoroughly about all details in the project and how I handle database, how I handle exceptions and many more.

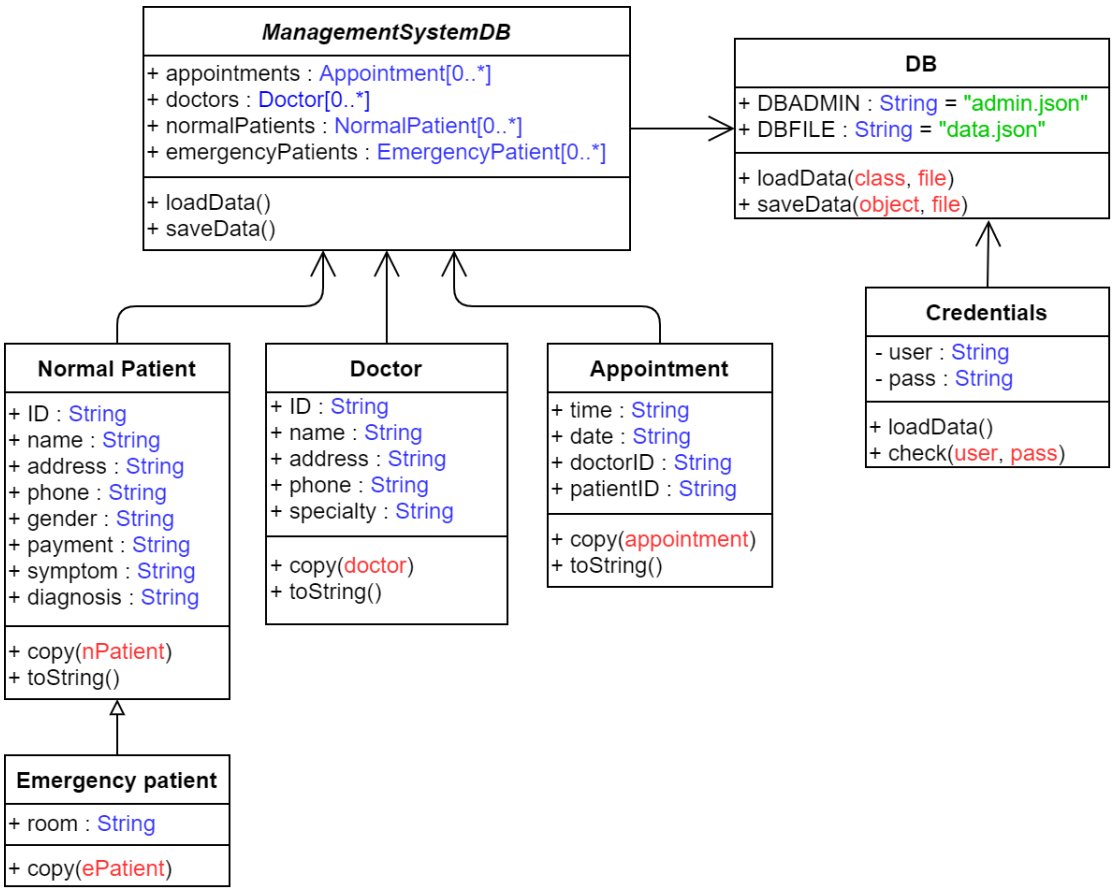
For the sake of simplicity, I did not implement a logging system. I really wanted to implement a multi-user software and store logs of who and what changes inside the program. But time was limited to try and implement a working logging system and it is out of the course scope as well.

2.0 UML Design

I will present a **database UML**, as well as a **GUI UML**.

2.1 Database UML

Let us start with this design for how data is stored and handled inside my program.



As seen in the figure, there is a management system class which stores all data. Let us begin with the DB class which has two files: DBADMIN (**admin credentials**) and DBFILE (**database**).

DB class capabilities:

- load data from **DBFILE** into a **ManagementSystemDB** object
- save data from a **ManagementSystemDB** object into **DBFILE**.

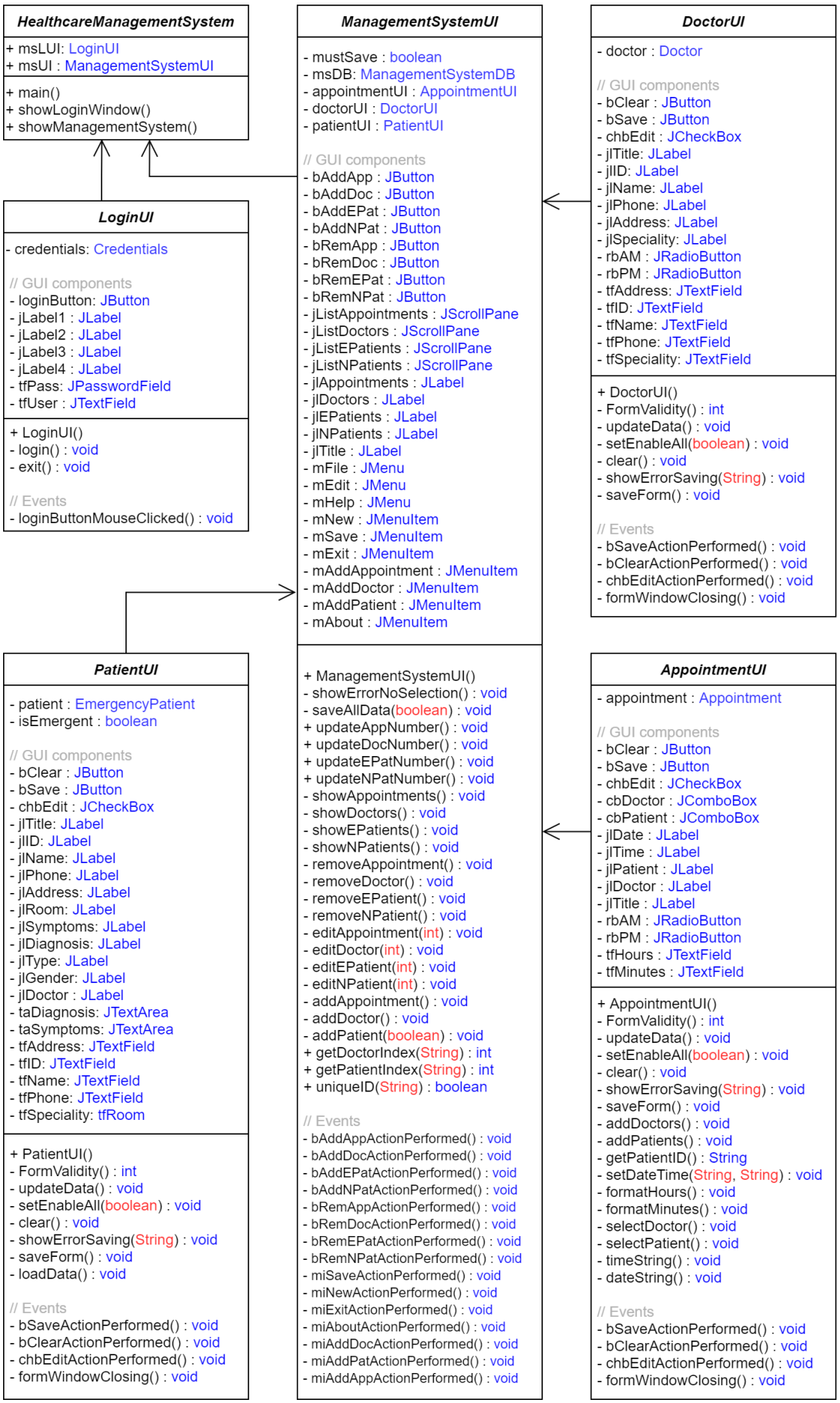
And "**no**" I did not create a backup system – although I would love to create one in case there is a fatal error in my program. And this is how I handle database in my program.

Credentials class handles administrator login to the system. It loads a username and a password locally and has a method to check if they match.

NormalPatient class, **EmergencyPatient** class, **Doctor** class and **Appointment** class are all according to the requirements in the PDF [Section 0.0 Introduction].

ManagementSystemDB class holds **ArrayLists** and one static object from it is used in all forms in the **UI**.

2.2 GUI UML


HealthcareManagementSystem
+ msLUI: LoginUI
+ msUI : ManagementSystemUI
+ main()
+ showLoginWindow()
+ showManagementSystem()
LoginUI
- credentials: Credentials
// GUI components
- loginButton: JButton
- jLabel1 : JLabel
- jLabel2 : JLabel
- jLabel3 : JLabel
- jLabel4 : JLabel
- tfPass: JPasswordField
- tfUser : JTextField
+ LoginUI()
- login() : void
- exit() : void
// Events
- loginButtonMouseClicked() : void
ManagementSystemUI
- mustSave : boolean
- msDB: ManagementSystemDB
- appointmentUI : AppointmentUI
- doctorUI : DoctorUI
- patientUI : PatientUI
// GUI components
- bAddApp : JButton
- bAddDoc : JButton
- bAddEPat : JButton
- bAddNPat : JButton
- bRemApp : JButton
- bRemDoc : JButton
- bRemEPat : JButton
- bRemNPat : JButton
- jListAppointments : JScrollPane
- jListDoctors : JScrollPane
- jListEPatients : JScrollPane
- jListNPatients : JScrollPane
- jlAppointments : JLabel
- jlDoctors : JLabel
- jlEPatients : JLabel
- jlNPatients : JLabel
- jlTitle : JLabel
- mFile : JMenu
- mEdit : JMenu
- mHelp : JMenu
- mNew : JMenuItem
- mSave : JMenuItem
- mExit : JMenuItem
- mAddAppointment : JMenuItem
- mAddDoctor : JMenuItem
- mAddPatient : JMenuItem
- mAbout : JMenuItem
+ ManagementSystemUI()
- showErrorNoSelection() : void
- saveAllData(boolean) : void
+ updateAppNumber() : void
+ updateDocNumber() : void
+ updateEPatNumber() : void
+ updateNPatNumber() : void
- showAppointments() : void
- showDoctors() : void
- showEPatients() : void
- showNPatients() : void
- removeAppointment() : void
- removeDoctor() : void
- removeEPatient() : void
- removeNPatient() : void
- editAppointment(int) : void
- editDoctor(int) : void
- editEPatient(int) : void
- editNPatient(int) : void
- addAppointment() : void
- addDoctor() : void
- addPatient(boolean) : void
+ getDoctorIndex(String) : int
+ getPatientIndex(String) : int
+ uniqueID(String) : boolean
// Events
- bAddAppActionPerformed() : void
- bAddDocActionPerformed() : void
- bAddEPatActionPerformed() : void
- bAddNPatActionPerformed() : void
- bRemAppActionPerformed() : void
- bRemDocActionPerformed() : void
- bRemEPatActionPerformed() : void
- bRemNPatActionPerformed() : void
- miSaveActionPerformed() : void
- miNewActionPerformed() : void
- miExitActionPerformed() : void
- miAboutActionPerformed() : void
- miAddDocActionPerformed() : void
- miAddPatActionPerformed() : void
- miAddAppActionPerformed() : void
DoctorUI
- doctor : Doctor
// GUI components
- bClear : JButton
- bSave : JButton
- chbEdit : JCheckBox
- jlTitle: JLabel
- jlID: JLabel
- jlName: JLabel
- jlPhone: JLabel
- jlAddress: JLabel
- jlSpeciality: JLabel
- rbAM : JRadioButton
- rbPM : JRadioButton
- tfAddress: JTextField
- tfID: JTextField
- tfName: JTextField
- tfPhone: JTextField
- tfSpeciality: JTextField
+ DoctorUI()
- FormValidity() : int
- updateData() : void
- setEnableAll(boolean) : void
- clear() : void
- showErrorSaving(String) : void
- saveForm() : void
// Events
- bSaveActionPerformed() : void
- bClearActionPerformed() : void
- chbEditActionPerformed() : void
- formWindowClosing() : void
PatientUI
- patient : EmergencyPatient
- isEmergent : boolean
// GUI components
- bClear : JButton
- bSave : JButton
- chbEdit : JCheckBox
- jlTitle: JLabel
- jlID: JLabel
- jlName: JLabel
- jlPhone: JLabel
- jlAddress: JLabel
- jlRoom: JLabel
- jlSymptoms: JLabel
- jlDiagnosis: JLabel
- jlType: JLabel
- jlGender: JLabel
- jlDoctor : JLabel
- taDiagnosis: JTextArea
- taSymptoms: JTextArea
- tfAddress: JTextField
- tfID: JTextField
- tfName: JTextField
- tfPhone: JTextField
- tfSpeciality: tfRoom
+ PatientUI()
- FormValidity() : int
- updateData() : void
- setEnableAll(boolean) : void
- clear() : void
- showErrorSaving(String) : void
- saveForm() : void
- loadData() : void
// Events
- bSaveActionPerformed() : void
- bClearActionPerformed() : void
- chbEditActionPerformed() : void
- formWindowClosing() : void
AppointmentUI
- appointment : Appointment
// GUI components
- bClear : JButton
- bSave : JButton
- chbEdit : JCheckBox
- cbDoctor : JComboBox
- cbPatient : JComboBox
- jlDate : JLabel
- jlTime : JLabel
- jlPatient : JLabel
- jlDoctor : JLabel
- jlTitle : JLabel
- rbAM : JRadioButton
- rbPM : JRadioButton
- tfHours : JTextField
- tfMinutes : JTextField
+ AppointmentUI()
- FormValidity() : int
- updateData() : void
- setEnableAll(boolean) : void
- clear() : void
- showErrorSaving(String) : void
- saveForm() : void
- addDoctors() : void
- addPatients() : void
- getPatientID() : String
- setDateTime(String, String) : void
- formatHours() : void
- formatMinutes() : void
- selectDoctor() : void
- selectPatient() : void
- timeString() : void
- dateString() : void
// Events
- bSaveActionPerformed() : void
- bClearActionPerformed() : void
- chbEditActionPerformed() : void
- formWindowClosing() : void

3.0 Used Libraries

I used many java libraries (included inside Netbeans):

- **java.util.ArrayList** (Array List)
- **javax.swing** (Swing used for GUI)

I used two libraries from the internet as well:

- **com.google.gson.Gson** (Gson reader & Gson writer)
- **com.toedter.calendar.JDateChooser** (Swing date chooser)

Note: I used Gson library to create the **DB class** which handles database.

4.0 Used Ready-made Functions

I do not use any ready-made functions in the project.

5.0 File Hierarchy

This tables shows the core files to that the program is made of.

Database Files
Appointment.java
Credentials.java
DB.java
Doctor.java
EmergencyPatient.java
HealthcareManagementSystem.java
ManagementSystem.java
NormalPatient.java

This table shows GUI code and resources (images)

GUI Files	GUI Images
AppointmentUI.java	**img_about.png**
DoctorUI.java	**img_appointment.png**
LoginUI.java	**img_doctor.png**
ManagementSystemUI.java	**img_exit.png**
PatientUI.java	**img_new.png**
	img_patient.png
	img_save.png
	login_image.png

Unfortunately, I programmed in the GUI directly instead of programming in another abstracted layer. By doing so, it will allow me (if I wanted to) to switch from my current GUI library quickly - in my case swing - to another. It also increases **flexibility**, **maintainability**, and **scalability**.

6.0 Copied Code

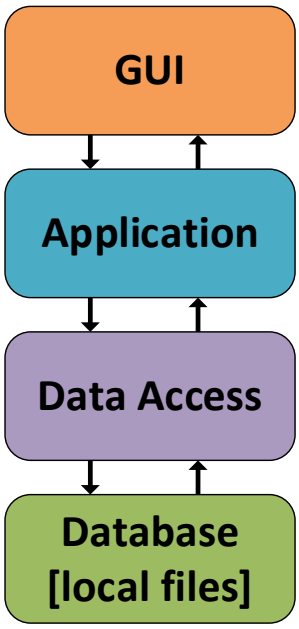
I did not copy and paste code into my program. I search for some functionalities in Swing and then I use them as it fits my program and as it requires.

7.0 Explanation

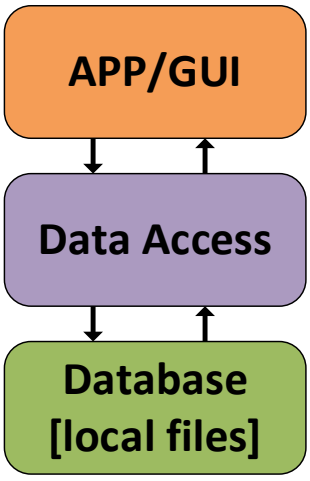
I will try my best to explain most of the code. I added many small tweaks to my software to make it behave like normal programs.

I will talk first about layered architecture and how I could have made my code cleaner, simpler, and more flexible.

This how an **ideal application layers** look like:



And this is how I programmed the application into my GUI directly. This is considered a bad approach in programming. It works fine, but not good for a team and splitting tasks among its members.



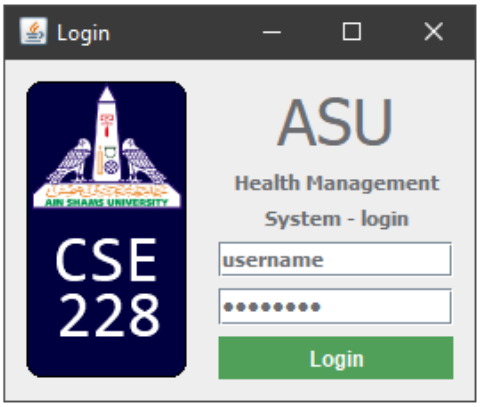
7.1 GUI Frames

Missing features:

- Logging System
- Database backing up system

7.1.1 LoginUI

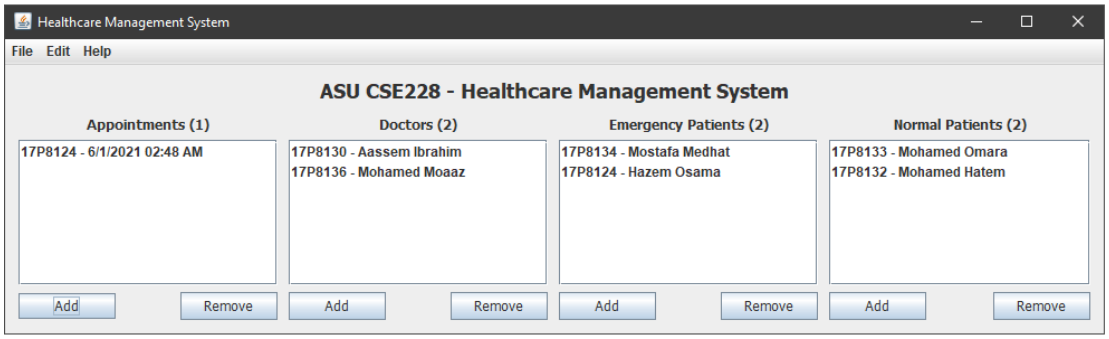
This window is the start of my program. If we entered the stored username and password in "**admin.json**", then main window opens. If the file is not found, then the default username and password are going to be "**admin**" and "**admin**".



7.1.1.1 LoginUI Missing features

- Change password
- Forgot password
- Create new user

7.1.2 ManagementSystemUI

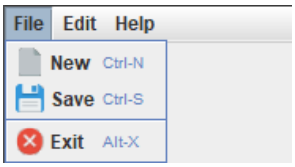


It loads "**data.json**" immediately after successful login. These are my friends, and I am included as a doctor. Each category has "Add" and "Remove" buttons that do what is written on them.

7.1.2.1 ManagementSystemUI Main Features

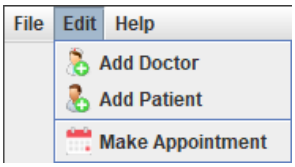
- Double clicking an item opens a window to **show/edit** item's data.
- Each category has a counter right beside their names and **refreshes immediately after any changes**.
- Ask user to **confirm before removal of an item**.
- Ask user again to **confirm before removal of linked items**.
- Removing **a doctor that has appointments** results to an undefined appointment which is displayed by "[?]" in the beginning of appointments name. Example: 17P8124 - 6/1/2021 01:25 AM → [?] 17P8124 - 6/1/2021 01:25 AM
- Removing **a patient that has appointments** results to the removal of all undefined appointments resulted from removing that patient; **as there is no need to have an appointment with no selected patient**. But in case of a doctor, the patient still needs an appointment but maybe with another doctor selecte.
- Adding any will result in adding an empty entry (just visually).
- Menubar that do special functionality as well as add buttons.

7.1.2.2 File Menu Functionalities



- New (clears all categories)
- Save (saves data to file locally)
- Exit (exit and save prompt appears if any changes)

7.1.2.3 Edit Menu Functionalities

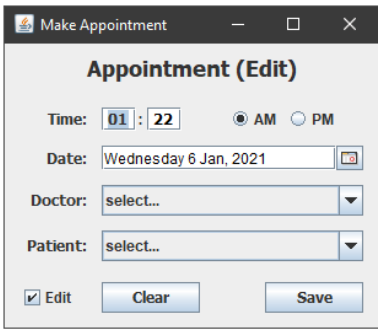


- Add Doctor
- Add Patient (add normal patient)
- Make Appointment

7.1.2.4 Help Menu Functionalities

- About (show small message)
- Github (open Github link) [I did not create it yet]

7.1.3 AppointmentUI

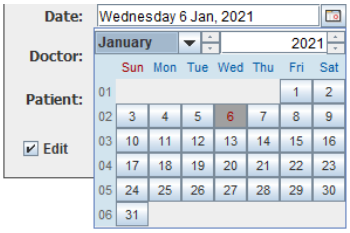


This window deals with setting an appointment and it is a link between a doctor and a patient.

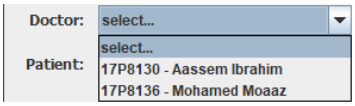
7.1.3.1 AppointmentUI Features

- Clear button resets everything.
- Edit checkbox is not check if we open an existing item and is checked if we created a new appointment. Also, the title label is linked with edit checkbox between "Edit/View only".
- Save button has many checks to make and all errors are defined and pop up a message with info. Everything is well defined inside this window.

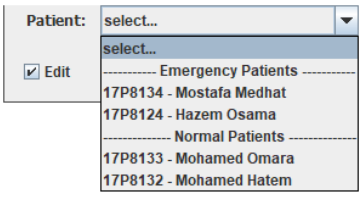
7.1.3.2 AppointmentUI Elements



JDateChooser

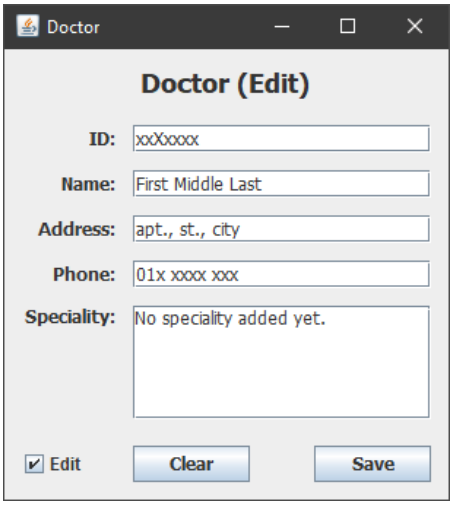


Doctor Selector



Patient Selector

7.1.4 DoctorUI



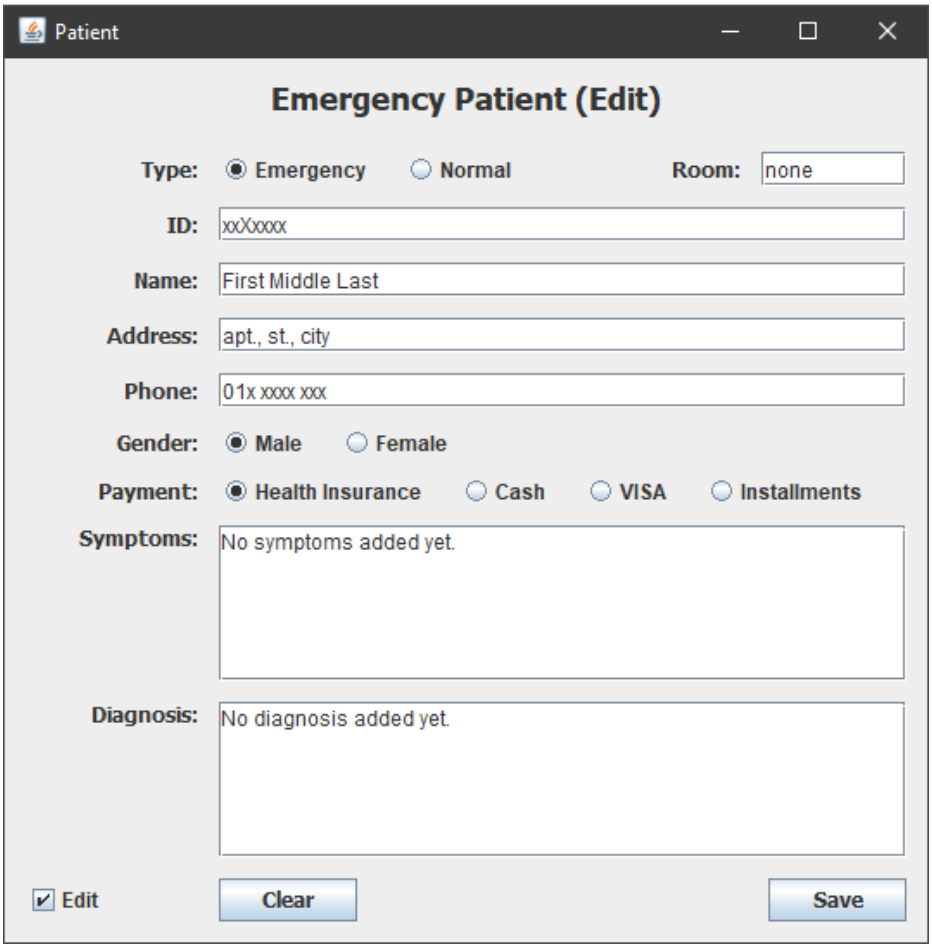
This window is made to straight forward filling up these **JTextFields**.

7.1.4.1 DoctorUI Features

- Clear (same as previous section)
- Save (same as previous section)
- Edit (same as previous section)

7.1.5 PatientUI

This window is for **normal** and **emergency patients**. When "Emergency" is selected, room is enabled and can have an input. When "Normal" is selected room is disabled.



7.1.5.1 PatientUI Features

- Clear (same as previous section)
- Save (same as previous section)
- Edit (same as previous section)
- **"Type" radio button group** ("Emergency" and "Normal") has the ability to transfer the patient data (inside **msDB**) from "ePatients" to "nPatients" or vice versa immediately [in data access layer].

7.1.5.2 PatientUI Missing Features

- Switching from emergency to normal needs a confirm prompt first.
- Clear does not check for ID and remove linked appointments as remove button in the ManagementSystemUI.

7.2 Code Explanation

I really do not know what "code explanation" is needed in this section. I will provide code explanation in the videos. The code is very big due to GUI generated lines of code.

HealthcareManagementSystem.java

```
1  package healthcaremanagementsystem;
2
3  public class HealthcareManagementSystem {
4      static LoginUI msLUI;                // LoginUI object
5      static ManagementSystemUI msUI;      // ManagementSystemUI object
6
7      public static void main(String[] args) {
8          showLoginWindow();               // Show login window
9      }
10
11     public static void showLoginWindow() {
12         msLUI = new LoginUI();
13     }
14
15     public static void showManagementSystem() {
16         msUI = new ManagementSystemUI();
17     }
18 }
```

This is my "main" and as shown I am calling "showLoginWindow" method which do as the name suggests.

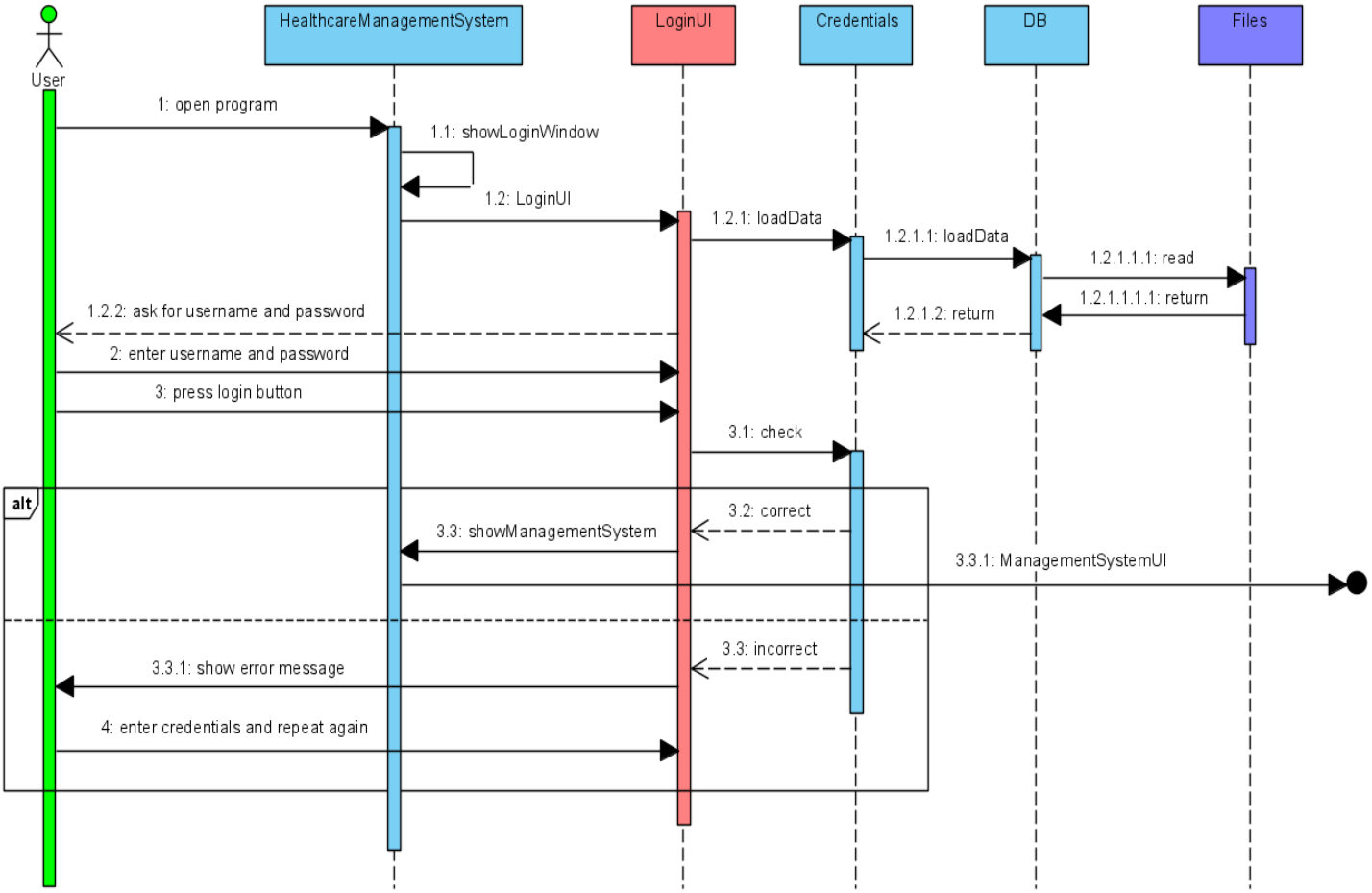
The Functions are static as I am calling "showManagementSystem" from outside this file scope (in LoginUI after entering correct credentials to login in).

Also, my "LoginUI" and "ManagementSystemUI" objects are static; as I will only have only one of each and I will use them in code later if needed.

7.3 UML Explanation

Note: I have already explained in the UML design section; to avoid repetition and to make my report cleaner and shorter.

I really like to present a simple sequence diagram that explains the first user interactions with my program.



This **sequence diagram** shows how the program flows starting from user at login until it reaches the main management system window that shows all 4 categories of main classes.

I wanted to create the rest of the sequence diagrams of my program, but time is not enough, and the program has about 10 to 15 diagrams in my mind that I want to share. Also, they are not a requirement in the project.

It makes more sense than writing many lines describing how the process goes.

8.0 Exceptions

I handled these types of exceptions:

- **ArrayIndexOutOfBoundsException** (important when needed)
- **ParseException** (in my program, I am parsing string into datetime)
- **FileNotFoundException** (if files not found on disk when read/write)
- **IOException** (input/output exception, usually at writing to file)

Always try and catch "**ArrayIndexOutOfBoundsException**" when accessing array lists in an unusual behavior. A bug-free software is very important for the clients.

Always try and catch "**ParseException**" while parsing. It is a good programming practice.